Advanced Astronomical Observations 2021
Session 10: Centroid and PSF Fitting

Kinoshita Daisuke

16 April 2021
publicly accessible version

About this file...

- Important information about this file
  - The author of this file is Kinoshita Daisuke.
  - The original version of this file was used for the course "Advanced Astronomical Observations" (course ID: AS6005) offered at Institute of Astronomy, National Central University from February 2021 to June 2021.
  - The file is provided in the hope that it will be useful, but there is no guarantee for the correctness. Use this file at your own risk.
  - If you are willing to use this file for your study, please feel free to use. I'll be very happy to receive feedback from you.
  - If you are willing to use this file for your teaching, please contact to Kinoshita Daisuke. When you use this file partly or entirely, please mention clearly that the author of the original version is Kinoshita Daisuke. Please help me to improve the contents of this file by sending your feedback.
  - Contact address: `https://www.instagram.com/daisuke23888/`

For this session, we try centroid measurements and PSF (Point-Spread Function) fitting of stellar images.

1 Photutils package

For this session, `photutils` package is needed. Visit following websites to learn about `photutils`.

- photutils
  - `https://photutils.readthedocs.io/` (Fig. 1)
  - `https://pypi.org/project/photutils/`
  - `https://github.com/astropy/photutils`

Try following to check whether you have `photutils` package installed on your computer. If you can import `photutils` package successfully, then you have `photutils` package properly installed on your computer.

```
% python3.9
Python 3.9.2 (default, Feb 21 2021, 12:39:42)
[GCC 7.5.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import photutils
>>>
```

If you see an error message like below, then you do not have `photutils` package on your computer.

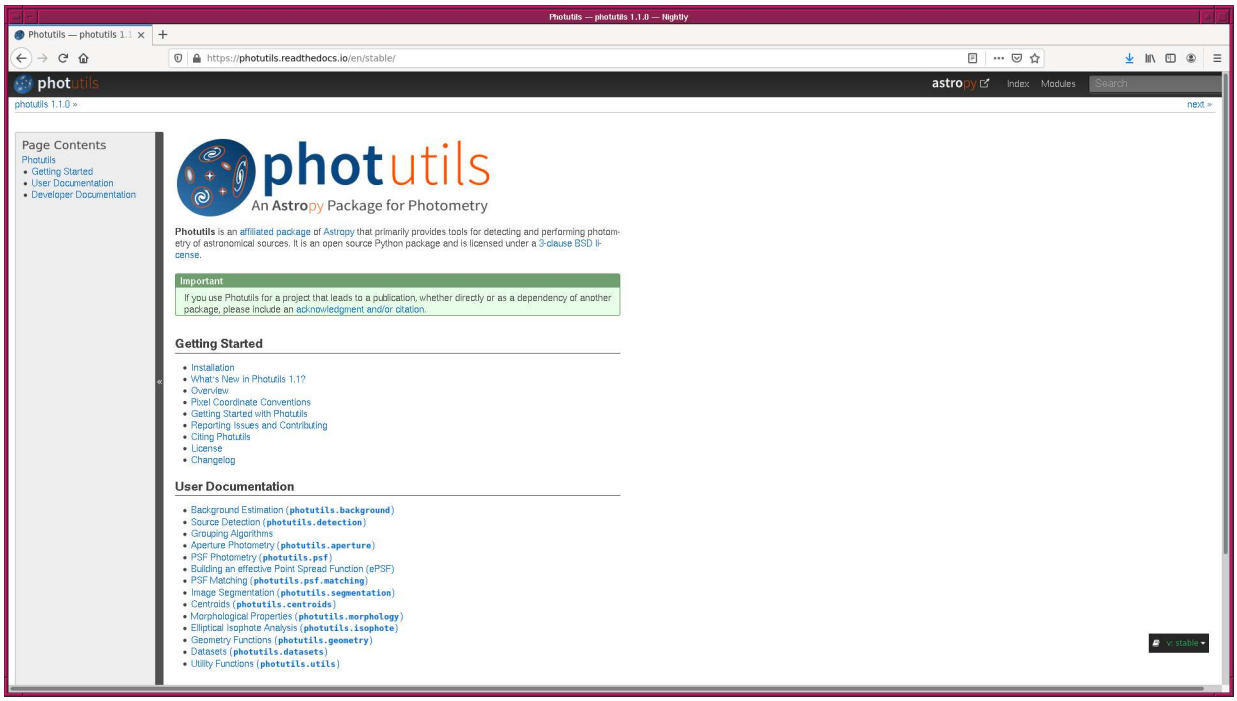

Figure 1: The official website of `photutils` package at `https://photutils.readthedocs.io/`.

```
% python3.9
Python 3.9.2 (default, Mar 27 2021, 17:26:25)
[GCC 7.5.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import photutils
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'photutils'
>>>
```

If you do not have `photutils` pakcage, visit the official website of `photutils` package (Fig. 2), read the installation instruction, and install the package on your computer.

2 Generating synthetic images

Use `photutils.datasets` to generate synthetic images.

2.1 Making a background image

Make a Python script to generate an image simulating sky background.

Python Code 1: ao2021_s10_01.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys
```

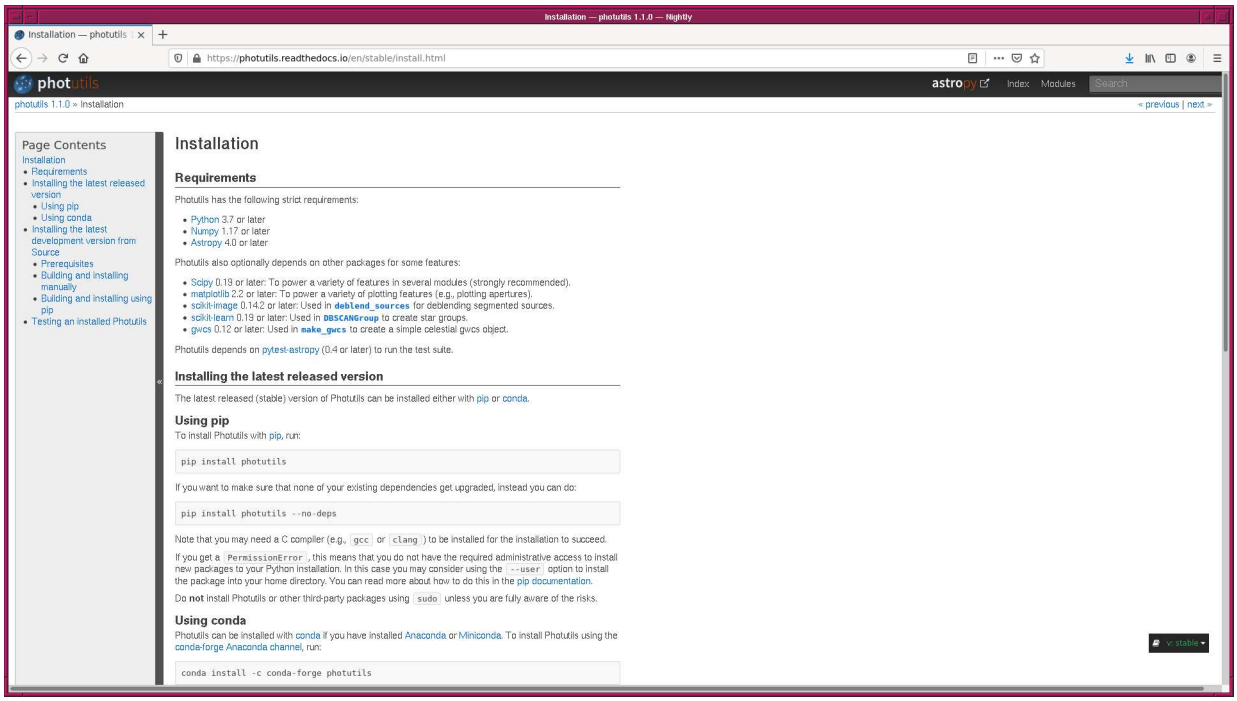

Figure 2: The installation instruction and description of the requirements at the official website of `photutils` package at `https://photutils.readthedocs.io/en/stable/install.html`.

```
# importing astropy module
import astropy.io.fits

# importing photutils module
import photutils.datasets

# constructing parser object
desc = 'generating a synthetic image'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-b', '--background', type=float, default=1000.0, \
                     help='background level')
parser.add_argument ('-s', '--sigma', type=float, default=10.0, \
                     help='noise level')
parser.add_argument ('-x', '--xsize', type=int, default=512, \
                     help='image size in x-axis')
parser.add_argument ('-y', '--ysize', type=int, default=512, \
                     help='image size in y-axis')
parser.add_argument ('-o', '--output', default='', \
                     help='output file name')

# command-line argument analysis
args = parser.parse_args ()

# input parameters
sky_background_level = args.background
noise_level          = args.sigma
image_size_x         = args.xsize
image_size_y         = args.ysize
file_output          = args.output
```

```
# checking output file name
if (file_output == ''):
    print ("You need to specify output file name.")
    sys.exit ()
if not (file_output[-5:] == '.fits'):
    print ("Output file must be a FITS file.")
    sys.exit ()

# image size
image_size = (image_size_x, image_size_y)

# generating sky background
image_background \
    = photutils.datasets.make_noise_image (image_size, \
                                           distribution='gaussian', \
                                           mean=sky_background_level, \
                                           stddev=noise_level)

# writing a FITS file
astropy.io.fits.writeto (file_output, image_background)
```

Execute the script and generate a synthetic image.

```
% chmod a+x ao2021_s10_01.py
% ./ao2021_s10_01.py -h
usage: ao2021_s10_01.py [-h] [-b BACKGROUND] [-s SIGMA] [-x XSIZE] [-y YSIZE]
                        [-o OUTPUT]

generating a synthetic image

optional arguments:
  -h, --help            show this help message and exit
  -b BACKGROUND, --background BACKGROUND
                        background level
  -s SIGMA, --sigma SIGMA
                        noise level
  -x XSIZE, --xsize XSIZE
                        image size in x-axis
  -y YSIZE, --ysize YSIZE
                        image size in y-axis
  -o OUTPUT, --output OUTPUT
                        output file name

% ./ao2021_s10_01.py -b 2000 -s 30 -x 1024 -y 1024 -o synthetic_00.fits
% ls -l synthetic_00.fits
-rw-r--r--  1 daisuke  taiwan  8392320 Apr 15 21:00 synthetic_00.fits
```

2.2 Examining pixel values

Make a Python script to examine pixel values of generated synthetic image.

Python Code 2: ao2021_s10_02.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse
```

```
# importing numpy module
import numpy
import numpy.ma

# importing astropy module
import astropy
import astropy.io.fits

# construction pf parser object
desc = 'calculating statistical information of FITS files'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
list_rejection = ['NONE', 'sigclip']
parser.add_argument ('-r', '--rejection', default='NONE', \
                     choices=list_rejection, \
                     help='rejection algorithm (default: NONE)')
parser.add_argument ('-t', '--threshold', type=float, default=4.0, \
                     help='threshold for sigma clipping (default: 4.0)')
parser.add_argument ('-n', '--maxiters', type=int, default=10, \
                     help='maximum number of iterations (default: 10)')
parser.add_argument ('files', nargs='+', help='FITS files')

# command-line argument analysis
args = parser.parse_args ()

# input parameters
rejection  = args.rejection
threshold  = args.threshold
maxiters   = args.maxiters
list_files = args.files

# printing information
print ("# Input parameters")
print ("#   rejection algorithm = %s" % rejection)
if not (rejection == 'NONE'):
    print ("#   threshold of sigma-clipping = %f" % threshold)
    print ("#   maximum number of iterations = %d" % maxiters)

# printing header
print ("#")
print ("# %-25s %8s %8s %8s %7s %8s %8s" \
       % ('file name', 'npix', 'mean', 'median', 'stddev', 'min', 'max') )
print ("#")

# scanning files
for file_fits in list_files:
    # if the file is not a FITS file, then skip
    if not (file_fits[-5:] == '.fits'):
        continue

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()
```

```
# opening FITS file
hdu_list = astropy.io.fits.open (file_fits)

# image of primary HDU
data0 = hdu_list[0].data

# closing FITS file
hdu_list.close ()

# calculations

# for no rejection algorithm
if (rejection == 'NONE'):
    # making a masked array
    data1 = numpy.ma.array (data0, mask=False)
# for sigma clipping algorithm
elif (rejection == 'sigclip'):
    data1 = numpy.ma.array (data0, mask=False)
    # iterations
    for j in range (maxiters):
        # number of usable pixels of previous iterations
        npix_prev = len (numpy.ma.compressed (data1) )
        # calculation of median
        median = numpy.ma.median (data1)
        # calculation of standard deviation
        stddev = numpy.ma.std (data1)
        # lower threshold
        low = median - threshold * stddev
        # higher threshold
        high = median + threshold * stddev
        # making a mask
        mask = (data1 < low) | (data1 > high)
        # making a masked array
        data1 = numpy.ma.array (data0, mask=mask)
        # number of usable pixels
        npix_now = len (numpy.ma.compressed (data1) )
        # leaving the loop, if number of usable pixels do not change
        if (npix_now == npix_prev):
            break

# calculation of mean, median, stddev, min, and max
mean   = numpy.ma.mean (data1)
median = numpy.ma.median (data1)
stddev = numpy.ma.std (data1)
vmin   = numpy.ma.min (data1)
vmax   = numpy.ma.max (data1)

# number of pixels
npix = len (data1.compressed () )

# file name
filename = file_fits.split ('/') [-1]

# printing result
print ("%-27s %8d %8.2f %8.2f %7.2f %8.2f %8.2f" \
       % (filename, npix, mean, median, stddev, vmin, vmax) )
```

Run the script and check pixel values.

```
% chmod a+x ao2021_s10_02.py
% ./ao2021_s10_02.py -h
usage: ao2021_s10_02.py [-h] [-r {NONE,sigclip}] [-t THRESHOLD] [-n MAXITERS]
                        files [files ...]

calculating statistical information of FITS files

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -r {NONE,sigclip}, --rejection {NONE,sigclip}
                        rejection algorithm (default: NONE)
  -t THRESHOLD, --threshold THRESHOLD
                        threshold for sigma clipping (default: 4.0)
  -n MAXITERS, --maxiters MAXITERS
                        maximum number of iterations (default: 10)

% ./ao2021_s10_02.py synthetic_00.fits
# Input parameters
#   rejection algorithm = NONE
#
# file name                    npix     mean    median   stddev      min      max
#
synthetic_00.fits            1048576  2000.01  2000.00    29.98  1840.98  2143.05
```

The mean and stddev are ~ 2000 and ~ 30, respectively, as expected.

2.3 Visual inspection of the image

Use Ginga to show the image. (Fig. 3)

```
% ginga synthetic_00.fits
```

2.4 Generating a synthetic image with stars

Make a Python script to generate a synthetic image simulating sky background and stars.

Python Code 3: ao2021_s10_03.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy

# importing astropy module
import astropy.io.fits
import astropy.table
```

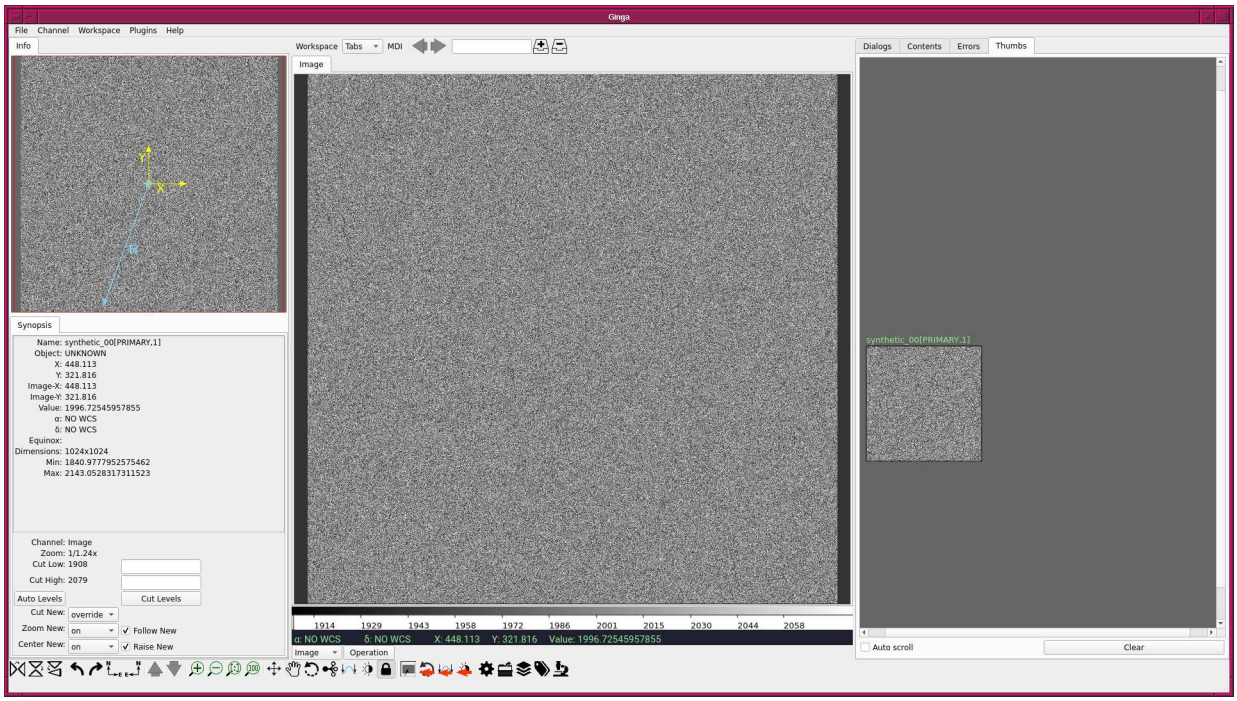

Figure 3: The synthetic image generated by photutils.datasets module.

```
# importing photutils module
import photutils.datasets

# constructing parser object
desc = 'generating a synthetic image'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-b', '--background', type=float, default=1000.0, \
                     help='background level (default: 1000)')
parser.add_argument ('-s', '--sigma', type=float, default=10.0, \
                     help='noise level (default: 10)')
parser.add_argument ('-n', '--nstar', type=int, default=10, \
                     help='number of stars to add (default: 10)')
parser.add_argument ('-f', '--flux', type=float, default=10000.0, \
                     help='maximum total flux of star (default: 10000)')
parser.add_argument ('-p', '--psf', type=float, default=5.0, \
                     help='FWHM of stellar radial profile (default: 5)')
parser.add_argument ('-x', '--xsize', type=int, default=512, \
                     help='image size in x-axis (default: 512)')
parser.add_argument ('-y', '--ysize', type=int, default=512, \
                     help='image size in y-axis (default: 512)')
parser.add_argument ('-o', '--output', default='', \
                     help='output file name')

# command-line argument analysis
args = parser.parse_args ()

# input parameters
sky_background_level = args.background
noise_level          = args.sigma
nstar                = args.nstar
flux_max             = args.flux
```

```
psf_fwhm              = args.psf
image_size_x          = args.xsize
image_size_y          = args.ysize
file_output           = args.output

# checking output file name
if (file_output == ''):
    print ("You need to specify output file name.")
    sys.exit ()
if not (file_output[-5:] == '.fits'):
    print ("Output file must be a FITS file.")
    sys.exit ()

# image size
image_size = (image_size_x, image_size_y)

# generating sky background
image_background \
    = photutils.datasets.make_noise_image (image_size, \
                                           distribution='gaussian', \
                                           mean=sky_background_level, \
                                           stddev=noise_level)

# making a source table
fwhm_sigma = 2.0 * numpy.sqrt (2.0 * numpy.log (2.0) )
source_table = astropy.table.Table ()
source_table['x_0']   = numpy.random.normal (image_size_x * 0.5, \
                                             image_size_x * 0.1, nstar)
source_table['y_0']   = numpy.random.normal (image_size_y * 0.5, \
                                             image_size_y * 0.1, nstar)
source_table['sigma'] = numpy.array ([psf_fwhm] * nstar) / fwhm_sigma
source_table['flux']  = numpy.random.uniform (flux_max * 0.1, flux_max, nstar)

# printing source table
print ("source_table:")
print (source_table)

# generating stars
image_star \
    = photutils.datasets.make_gaussian_prf_sources_image (image_size, \
                                                          source_table)

# making synthetic image
image = image_background + image_star

# writing a FITS file
astropy.io.fits.writeto (file_output, image)
```

Run the script and generate a synthetic image with stars.

```
% chmod a+x ao2021_s10_03.py
% ./ao2021_s10_03.py -h
usage: ao2021_s10_03.py [-h] [-b BACKGROUND] [-s SIGMA] [-n NSTAR] [-f FLUX]
                        [-p PSF] [-x XSIZE] [-y YSIZE] [-o OUTPUT]

generating a synthetic image

optional arguments:
```

```
  -h, --help            show this help message and exit
  -b BACKGROUND, --background BACKGROUND
                        background level (default: 1000)
  -s SIGMA, --sigma SIGMA
                        noise level (default: 10)
  -n NSTAR, --nstar NSTAR
                        number of stars to add (default: 10)
  -f FLUX, --flux FLUX  maximum total flux of star (default: 10000)
  -p PSF, --psf PSF     FWHM of stellar radial profile (default: 5)
  -x XSIZE, --xsize XSIZE
                        image size in x-axis (default: 512)
  -y YSIZE, --ysize YSIZE
                        image size in y-axis (default: 512)
  -o OUTPUT, --output OUTPUT
                        output file name

% ./ao2021_s10_03.py -b 2000 -s 30 -n 500 -f 50000 -p 5 -x 1024 -y 1024 \
? -o synthetic_01.fits
source_table:
       x_0                 y_0                sigma                flux
------------------ ------------------ ------------------ ------------------
341.41762216200505 444.04146666173295 2.1233045007200477 14292.212998313778
  567.594960912384  642.0830565873991 2.1233045007200477  9359.595876245241
 604.4954241291257 404.37438166817566 2.1233045007200477 23139.502398112145
 301.2810032045135   587.853842459603 2.1233045007200477 31693.575429892448
394.33150270830737   528.485069306676 2.1233045007200477 38690.317185518514
420.94935971761777 466.76844675291835 2.1233045007200477 42734.012981805914
 475.2994285876409 428.40654235573083 2.1233045007200477 41614.102224308655
               ...                ...                ...                ...
386.00743904730155  492.6498443998063 2.1233045007200477  48922.04868459824
456.59727622416915  455.5450608668942 2.1233045007200477  41926.74965293204
 576.3641250616013  549.0464408738366 2.1233045007200477  43454.21056141639
 616.8031681447002  572.4571020935433 2.1233045007200477  29775.38602380078
 399.7648748733322  495.9909920982335 2.1233045007200477  40048.55192298604
402.58012934079875  476.3093464751816 2.1233045007200477 30309.564075062855
 497.0512691464913  637.3531460500942 2.1233045007200477   44629.3918843194
Length = 500 rows
% ls -l synthetic_01.fits
-rw-r--r--  1 daisuke  taiwan  8392320 Apr 15 21:53 synthetic_01.fits
```

2.5 Visual inspection of the image

Use Ginga to show the image. (Fig. 4)

```
% ginga synthetic_01.fits
```

3 Centroid measurements

To know the location of a star, centroid is often used.

3.1 Knowing a rough position of a star using Ginga

Use Ginga to know a rough position of a star.

1. Start Ginga.
2. Load the image "`synthetic_01.fits`".

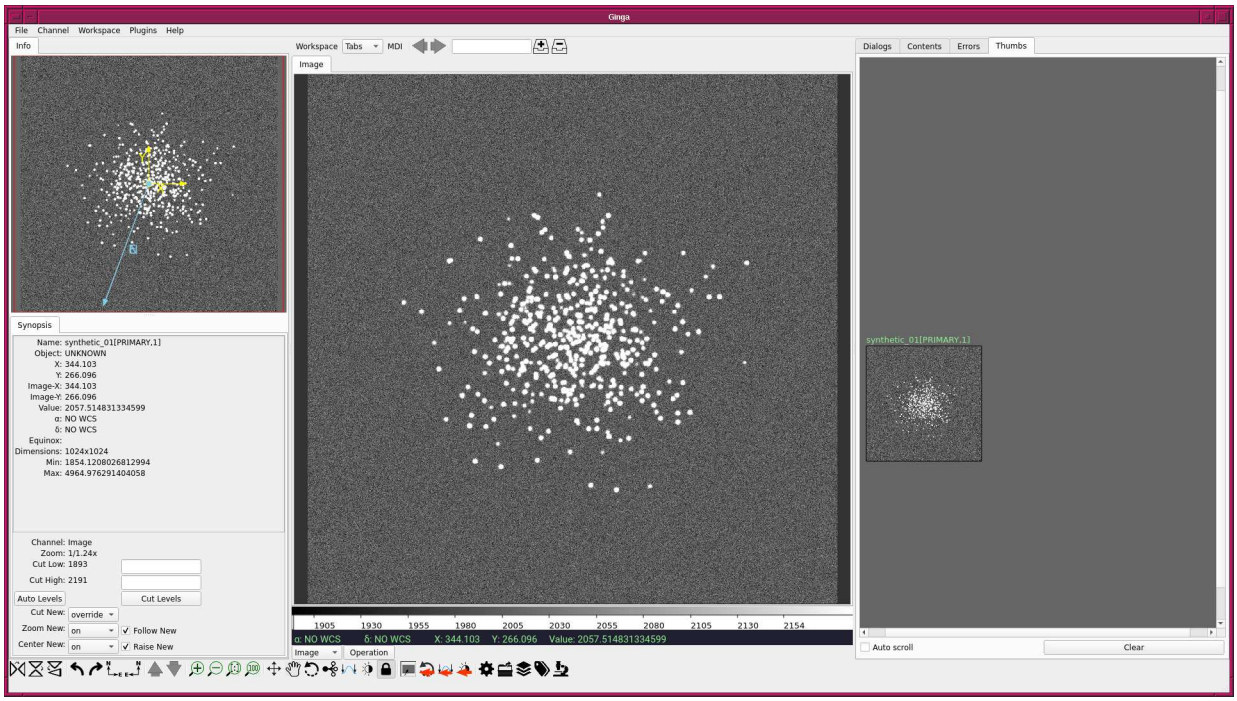

Figure 4: The synthetic image with stars generated by photutils.datasets module.

3. Pick a star for your centroid measurement. Choose a relatively isolated star.
4. Zoom in and pan to the star you have picked.
5. Move your mouse cursor to your target.
6. Read rough values of (x, y) coordinate of your target.

For example, a star at (x, y) $\sim$ (325, 274) is selected. (Fig. 5)

3.2 Centroid measurement using centre of mass

Make a Python script to carry out centroid measurement using centre of mass.

Python Code 4: ao2021_s10_04.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy

# importing astropy module
import astropy.io.fits

# importing photutils module
import photutils.centroids

# constructing parser object
desc = 'centroid measurement using centre of mass'
```

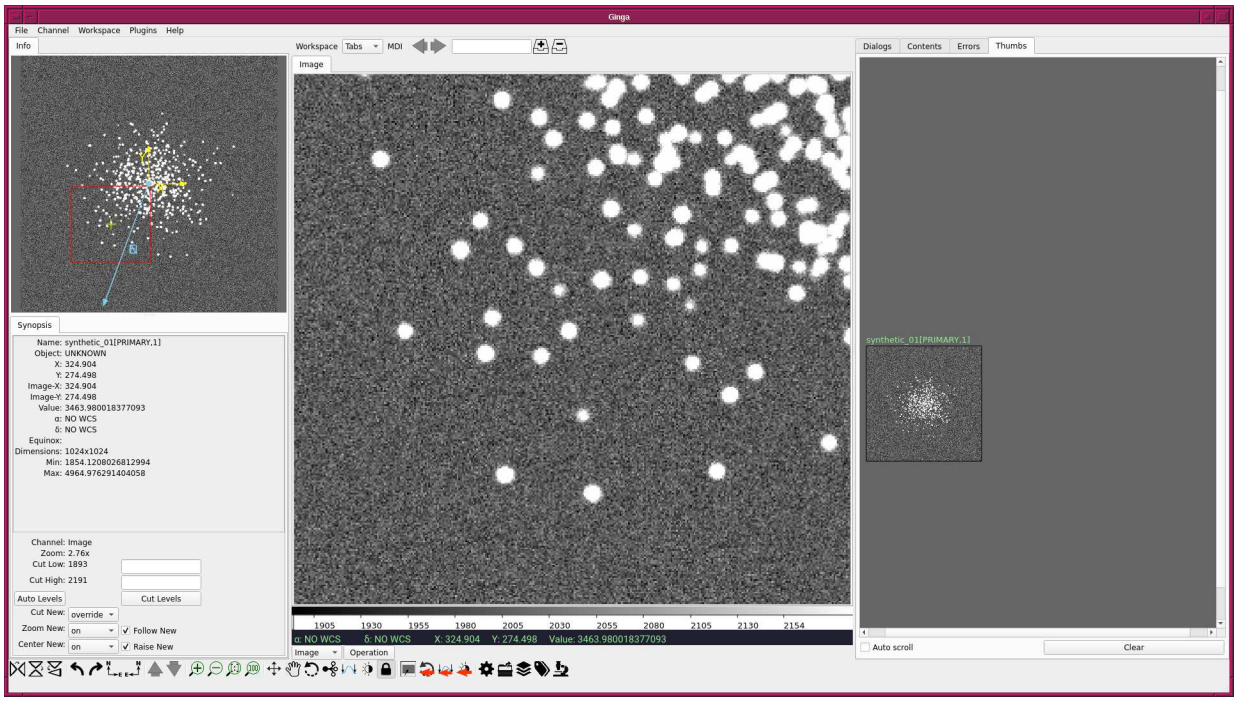

Figure 5: A rough position of a star is read using Ginga.

```
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-w', '--width', type=int, default=5, \
                     help='half-width of centroid calculation box (default: 5)')
parser.add_argument ('-x', '--xinit', type=int, default=-1, \
                     help='a rough x coordinate of target')
parser.add_argument ('-y', '--yinit', type=int, default=-1, \
                     help='a rough y coordinate of target')
parser.add_argument ('file', nargs=1, default='', help='input file name')

# command-line argument analysis
args = parser.parse_args ()

# input parameters
half_width = args.width
x_init     = args.xinit
y_init     = args.yinit
file_fits  = args.file[0]

# checking output file name
if (file_fits == ''):
    print ("You need to specify input file name.")
    sys.exit ()
if not (file_fits[-5:] == '.fits'):
    print ("Input file must be a FITS file.")
    sys.exit ()

# opening FITS file
hdu_list = astropy.io.fits.open (file_fits)

# reading FITS header
header = hdu_list[0].header
```

```
# image size
image_size_x = header['NAXIS1']
image_size_y = header['NAXIS2']

# checking x_init and y_init
if not ( (x_init >=0) and (x_init < image_size_x) ):
    print ("Input x_init value exceed image size.")
    sys.exit ()
if not ( (y_init >=0) and (y_init < image_size_y) ):
    print ("Input y_init value exceed image size.")
    sys.exit ()

# reading FITS image data
data = hdu_list[0].data

# closing FITS file
hdu_list.close ()

# region of calculation
x_min = x_init - half_width
x_max = x_init + half_width + 1
y_min = y_init - half_width
y_max = y_init + half_width + 1

# extraction of subframe for calculation
subframe = data[y_min:y_max, x_min:x_max]

# rough background subtraction
subframe -= numpy.median (subframe)

# centroid calculation
(x_centre, y_centre) = photutils.centroids.centroid_com (subframe)
x_centre += x_min
y_centre += y_min

# printing information
print ("#")
print ("# input file name = %s" % file_fits)
print ("# half-width of search box = %f" % half_width)
print ("# x_init = %f" % x_init)
print ("# y_init = %f" % y_init)
print ("#")

# printing result
print ("%f %f" % (x_centre, y_centre) )
```

Execute the script, and measure the location of the star.

```
% chmod a+x ao2021_s10_04.py
% ./ao2021_s10_04.py -h
usage: ao2021_s10_04.py [-h] [-w WIDTH] [-x XINIT] [-y YINIT] file

centroid measurement using centre of mass

positional arguments:
  file                  input file name
```

```
optional arguments:
  -h, --help            show this help message and exit
  -w WIDTH, --width WIDTH
                        half-width of centroid calculation box (default: 5)
  -x XINIT, --xinit XINIT
                        a rough x coordinate of target
  -y YINIT, --yinit YINIT
                        a rough y coordinate of target

% ./ao2021_s10_04.py -w 10 -x 325 -y 274 synthetic_01.fits
#
# input file name = synthetic_01.fits
# half-width of search box = 10.000000
# x_init = 325.000000
# y_init = 274.000000
#
322.529580 273.436223
```

3.3 Centroid measurement using 1D Gaussian fitting

Try 1D Gaussian fitting for centroid measurement.

Python Code 5: ao2021_s10_05.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy

# importing astropy module
import astropy.io.fits

# importing photutils module
import photutils.centroids

# constructing parser object
desc = 'centroid measurement using 1D Gaussian fitting'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-w', '--width', type=int, default=5, \
                     help='half-width of centroid calculation box (default: 5)')
parser.add_argument ('-x', '--xinit', type=int, default=-1, \
                     help='a rough x coordinate of target')
parser.add_argument ('-y', '--yinit', type=int, default=-1, \
                     help='a rough y coordinate of target')
parser.add_argument ('file', nargs=1, default='', help='input file name')

# command-line argument analysis
args = parser.parse_args ()

# input parameters
half_width = args.width
```

```
x_init      = args.xinit
y_init      = args.yinit
file_fits   = args.file[0]

# checking output file name
if (file_fits == ''):
    print ("You need to specify input file name.")
    sys.exit ()
if not (file_fits[-5:] == '.fits'):
    print ("Input file must be a FITS file.")
    sys.exit ()

# opening FITS file
hdu_list = astropy.io.fits.open (file_fits)

# reading FITS header
header = hdu_list[0].header

# image size
image_size_x = header['NAXIS1']
image_size_y = header['NAXIS2']

# checking x_init and y_init
if not ( (x_init >=0) and (x_init < image_size_x) ):
    print ("Input x_init value exceed image size.")
    sys.exit ()
if not ( (y_init >=0) and (y_init < image_size_y) ):
    print ("Input y_init value exceed image size.")
    sys.exit ()

# reading FITS image data
data = hdu_list[0].data

# closing FITS file
hdu_list.close ()

# region of calculation
x_min = x_init - half_width
x_max = x_init + half_width + 1
y_min = y_init - half_width
y_max = y_init + half_width + 1

# extraction of subframe for calculation
subframe = data[y_min:y_max, x_min:x_max]

# rough background subtraction
subframe -= numpy.median (subframe)

# centroid calculation
(x_centre, y_centre) = photutils.centroids.centroid_1dg (subframe)
x_centre += x_min
y_centre += y_min

# printing information
print ("#")
print ("# input file name = %s" % file_fits)
print ("# half-width of search box = %f" % half_width)
print ("# x_init = %f" % x_init)
print ("# y_init = %f" % y_init)
```

```
print ("#")

# printing result
print ("%f %f" % (x_centre, y_centre) )
```

Run the script.

```
% chmod a+x ao2021_s10_05.py
% ./ao2021_s10_05.py -h
usage: ao2021_s10_05.py [-h] [-w WIDTH] [-x XINIT] [-y YINIT] file

centroid measurement using 1D Gaussian fitting

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -w WIDTH, --width WIDTH
                        half-width of centroid calculation box (default: 5)
  -x XINIT, --xinit XINIT
                        a rough x coordinate of target
  -y YINIT, --yinit YINIT
                        a rough y coordinate of target

% ./ao2021_s10_05.py -w 10 -x 325 -y 274 synthetic_01.fits
#
# input file name = synthetic_01.fits
# half-width of search box = 10.000000
# x_init = 325.000000
# y_init = 274.000000
#
322.914793 273.573125
```

3.4 Centroid measurement using 2D Gaussian fitting

Try 2D Gaussian fitting for centroid measurement.

Python Code 6: ao2021_s10_06.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy

# importing astropy module
import astropy.io.fits

# importing photutils module
import photutils.centroids

# constructing parser object
```

```
desc = 'centroid measurement using 2D Gaussian fitting'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-w', '--width', type=int, default=5, \
                     help='half-width of centroid calculation box (default: 5)')
parser.add_argument ('-x', '--xinit', type=int, default=-1, \
                     help='a rough x coordinate of target')
parser.add_argument ('-y', '--yinit', type=int, default=-1, \
                     help='a rough y coordinate of target')
parser.add_argument ('file', nargs=1, default='', help='input file name')

# command-line argument analysis
args = parser.parse_args ()

# input parameters
half_width = args.width
x_init     = args.xinit
y_init     = args.yinit
file_fits  = args.file[0]

# checking output file name
if (file_fits == ''):
    print ("You need to specify input file name.")
    sys.exit ()
if not (file_fits[-5:] == '.fits'):
    print ("Input file must be a FITS file.")
    sys.exit ()

# opening FITS file
hdu_list = astropy.io.fits.open (file_fits)

# reading FITS header
header = hdu_list[0].header

# image size
image_size_x = header['NAXIS1']
image_size_y = header['NAXIS2']

# checking x_init and y_init
if not ( (x_init >=0) and (x_init < image_size_x) ):
    print ("Input x_init value exceed image size.")
    sys.exit ()
if not ( (y_init >=0) and (y_init < image_size_y) ):
    print ("Input y_init value exceed image size.")
    sys.exit ()

# reading FITS image data
data = hdu_list[0].data

# closing FITS file
hdu_list.close ()

# region of calculation
x_min = x_init - half_width
x_max = x_init + half_width + 1
y_min = y_init - half_width
y_max = y_init + half_width + 1
```

```
# extraction of subframe for calculation
subframe = data[y_min:y_max, x_min:x_max]

# rough background subtraction
subframe -= numpy.median (subframe)

# centroid calculation
(x_centre, y_centre) = photutils.centroids.centroid_2dg (subframe)
x_centre += x_min
y_centre += y_min

# printing information
print ("#")
print ("# input file name = %s" % file_fits)
print ("# half-width of search box = %f" % half_width)
print ("# x_init = %f" % x_init)
print ("# y_init = %f" % y_init)
print ("#")

# printing result
print ("%f %f" % (x_centre, y_centre) )
```

Run the script.

```
% chmod a+x ao2021_s10_06.py
% ./ao2021_s10_06.py -h
usage: ao2021_s10_06.py [-h] [-w WIDTH] [-x XINIT] [-y YINIT] file

centroid measurement using 2D Gaussian fitting

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -w WIDTH, --width WIDTH
                        half-width of centroid calculation box (default: 5)
  -x XINIT, --xinit XINIT
                        a rough x coordinate of target
  -y YINIT, --yinit YINIT
                        a rough y coordinate of target

% ./ao2021_s10_06.py -w 10 -x 325 -y 274 synthetic_01.fits
#
# input file name = synthetic_01.fits
# half-width of search box = 10.000000
# x_init = 325.000000
# y_init = 274.000000
#
322.928392 273.562487
```

3.5 Visualising result of centroid measurement

Make a Python script to carry out centroid measurement and visualising the result.

Python Code 7: ao2021_s10_07.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy

# importing astropy module
import astropy.io.fits

# importing photutils module
import photutils.centroids

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# constructing parser object
desc = 'centroid measurement'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
list_centroid = ['com', '1dg', '2dg']
parser.add_argument ('-c', '--centroid', choices=list_centroid, default='com', \
                     help='centroid measurement algorithm (default: com)')
parser.add_argument ('-o', '--output', default='centroid.png', \
                     help='output file name (default: centroid.png)')
parser.add_argument ('-w', '--width', type=int, default=5, \
                     help='half-width of centroid calculation box (default: 5)')
parser.add_argument ('-x', '--xinit', type=int, default=-1, \
                     help='a rough x coordinate of target')
parser.add_argument ('-y', '--yinit', type=int, default=-1, \
                     help='a rough y coordinate of target')
parser.add_argument ('file', nargs=1, default='', help='input file name')

# command-line argument analysis
args = parser.parse_args ()

# input parameters
centroid    = args.centroid
half_width  = args.width
x_init      = args.xinit
y_init      = args.yinit
file_fits   = args.file[0]
file_output = args.output

# checking input file name
if (file_fits == ''):
    print ("You need to specify input file name.")
    sys.exit ()
if not (file_fits[-5:] == '.fits'):
    print ("Input file must be a FITS file.")
    sys.exit ()

# checking output file name
```

```
if not ( (file_output[-4:] == '.eps') or (file_output[-4:] == '.pdf') \
         or (file_output[-4:] == '.png') or (file_output[-3:] == '.ps') ):
    print ("Output file must be EPS or PDF or PNG or PS.")
    sys.exit ()

# opening FITS file
hdu_list = astropy.io.fits.open (file_fits)

# reading FITS header
header = hdu_list[0].header

# image size
image_size_x = header['NAXIS1']
image_size_y = header['NAXIS2']

# checking x_init and y_init
if not ( (x_init >=0) and (x_init < image_size_x) ):
    print ("Input x_init value exceed image size.")
    sys.exit ()
if not ( (y_init >=0) and (y_init < image_size_y) ):
    print ("Input y_init value exceed image size.")
    sys.exit ()

# reading FITS image data
data = hdu_list[0].data

# closing FITS file
hdu_list.close ()

# region of calculation
x_min = x_init - half_width
x_max = x_init + half_width + 1
y_min = y_init - half_width
y_max = y_init + half_width + 1

# extraction of subframe for calculation
subframe = data[y_min:y_max, x_min:x_max]

# rough background subtraction
subframe -= numpy.median (subframe)

# centroid calculation
if (centroid == 'com'):
    (x_centre, y_centre) = photutils.centroids.centroid_com (subframe)
elif (centroid == '1dg'):
    (x_centre, y_centre) = photutils.centroids.centroid_1dg (subframe)
elif (centroid == '2dg'):
    (x_centre, y_centre) = photutils.centroids.centroid_2dg (subframe)
x_centre_sub = x_centre
y_centre_sub = y_centre
x_centre += x_min
y_centre += y_min

# printing information
print ("#")
print ("# input file name = %s" % file_fits)
print ("# half-width of search box = %f" % half_width)
print ("# x_init = %f" % x_init)
print ("# y_init = %f" % y_init)
```

```
print ("#")

# printing result
print ("%f %f" % (x_centre, y_centre) )

# making objects "fig" and "ax"
fig = matplotlib.figure.Figure ()
matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax = fig.add_subplot (111)

# axes
ax.set_xlabel ('X [pixel]')
ax.set_ylabel ('Y [pixel]')

# plotting image
im = ax.imshow (subframe, origin='lower')
fig.colorbar (im)
ax.plot (x_centre_sub, y_centre_sub, marker='+', color='red', markersize=10)

# saving file
fig.savefig (file_output, dpi=225)
```

Run the script and generate a PNG file.

```
% chmod a+x ao2021_s10_07.py
% ./ao2021_s10_07.py -h
usage: ao2021_s10_07.py [-h] [-c {com,1dg,2dg}] [-o OUTPUT] [-w WIDTH]
                        [-x XINIT] [-y YINIT]
                        file

centroid measurement

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -c {com,1dg,2dg}, --centroid {com,1dg,2dg}
                        centroid measurement algorithm (default: com)
  -o OUTPUT, --output OUTPUT
                        output file name (default: centroid.png)
  -w WIDTH, --width WIDTH
                        half-width of centroid calculation box (default: 5)
  -x XINIT, --xinit XINIT
                        a rough x coordinate of target
  -y YINIT, --yinit YINIT
                        a rough y coordinate of target

% ./ao2021_s10_07.py -c 2dg -o synthetic_01_2dg.png -w 10 -x 325 -y 274 \
? synthetic_01.fits
#
# input file name = synthetic_01.fits
# half-width of search box = 10.000000
# x_init = 325.000000
# y_init = 274.000000
#
322.928392 273.562487
% ls -l synthetic_01_2dg.png
```

```
-rw-r--r--  1 daisuke  taiwan  55301 Apr 15 23:24 synthetic_01_2dg.png
```

Show the PNG file. (Fig. 6)

```
% feh -dF synthetic_01_2dg.png
```

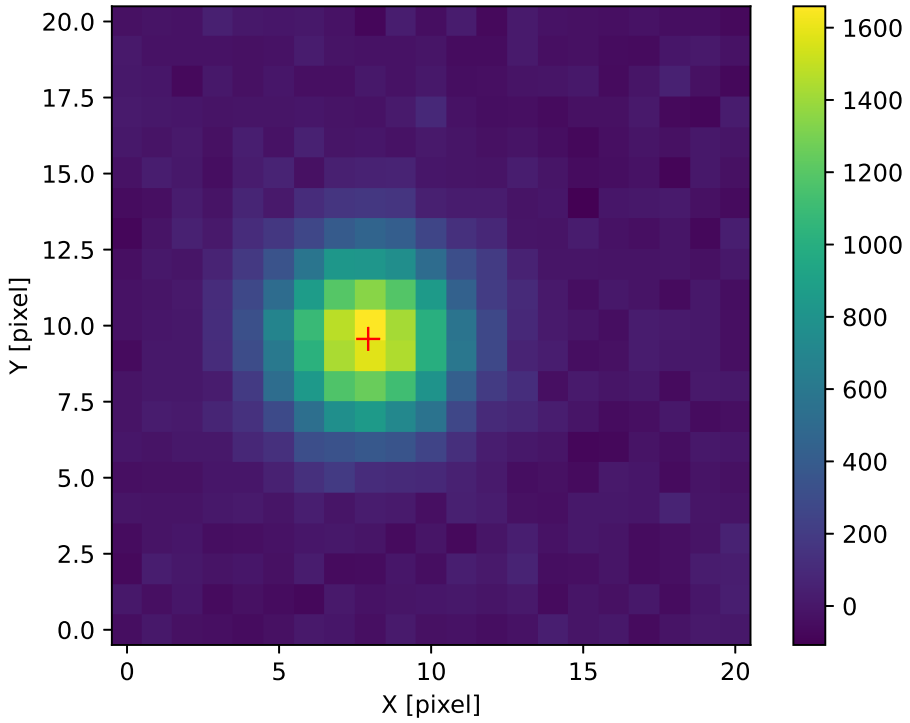

Figure 6: Result of centroid measurement using 2D Gaussian fitting.

4 PSF fitting

Try PSF fitting using `astropy.modeling` module.

4.1 PSF fitting using 2D Gaussian function

Make a Python script to carry out PSF fitting using 2D Gaussian function.

Python Code 8: ao2021_s10_08.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
```

```
import numpy

# importing astropy module
import astropy.io.fits
import astropy.modeling

# importing photutils module
import photutils.centroids

# constructing parser object
desc = 'centroid measurement'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-w', '--width', type=int, default=5, \
                     help='half-width of centroid calculation box (default: 5)')
parser.add_argument ('-x', '--xinit', type=int, default=-1, \
                     help='a rough x coordinate of target')
parser.add_argument ('-y', '--yinit', type=int, default=-1, \
                     help='a rough y coordinate of target')
parser.add_argument ('file', nargs=1, default='', help='input file name')

# command-line argument analysis
args = parser.parse_args ()

# input parameters
half_width  = args.width
x_init      = args.xinit
y_init      = args.yinit
file_fits   = args.file[0]

# checking input file name
if (file_fits == ''):
    print ("You need to specify input file name.")
    sys.exit ()
if not (file_fits[-5:] == '.fits'):
    print ("Input file must be a FITS file.")
    sys.exit ()

# opening FITS file
hdu_list = astropy.io.fits.open (file_fits)

# reading FITS header
header = hdu_list[0].header

# image size
image_size_x = header['NAXIS1']
image_size_y = header['NAXIS2']

# checking x_init and y_init
if not ( (x_init >=0) and (x_init < image_size_x) ):
    print ("Input x_init value exceed image size.")
    sys.exit ()
if not ( (y_init >=0) and (y_init < image_size_y) ):
    print ("Input y_init value exceed image size.")
    sys.exit ()

# reading FITS image data
data = hdu_list[0].data
```

```
# closing FITS file
hdu_list.close ()

# region of calculation
x_min = x_init - half_width
x_max = x_init + half_width + 1
y_min = y_init - half_width
y_max = y_init + half_width + 1

# extraction of subframe for calculation
subframe = data[y_min:y_max, x_min:x_max]

# rough background subtraction
subframe -= numpy.median (subframe)

# centroid measurement
(xc_com, yc_com) = photutils.centroids.centroid_com (subframe)

# PSF fitting
subframe_y, subframe_x = numpy.indices (subframe.shape)
psf_init = astropy.modeling.models.Gaussian2D (x_mean=xc_com, y_mean=yc_com)
fit = astropy.modeling.fitting.LevMarLSQFitter ()
psf_fitted = fit (psf_init, subframe_x, subframe_y, subframe)

# result of fitting
x_centre     = psf_fitted.x_mean.value
y_centre     = psf_fitted.y_mean.value
x_centre_sub = x_centre
y_centre_sub = y_centre
x_centre    += x_min
y_centre    += y_min
x_fwhm       = psf_fitted.x_fwhm
y_fwhm       = psf_fitted.y_fwhm
fwhm         = (x_fwhm + y_fwhm) / 2.0
amplitude    = psf_fitted.amplitude.value
theta        = psf_fitted.theta.value

# printing information
print ("#")
print ("# input file name = %s" % file_fits)
print ("# half-width of search box = %f" % half_width)
print ("# x_init = %f" % x_init)
print ("# y_init = %f" % y_init)
print ("#")
print ("# result")
print ("#  x_centre  = %f" % x_centre)
print ("#  y_centre  = %f" % y_centre)
print ("#  x_fwhm    = %f" % x_fwhm)
print ("#  y_fwhm    = %f" % y_fwhm)
print ("#  amplitude = %f" % amplitude)
print ("#  theta     = %f" % theta)
print ("#")

# printing result
print ("# x_fwhm, y_fwhm, fwhm")
print ("%f %f %f" % (x_fwhm, y_fwhm, fwhm) )
```

Execute the script, and measure FWHM (Full-Width at Half-Maximum) of stellar profile.

```
% chmod a+x ao2021_s10_08.py
% ./ao2021_s10_08.py -h
usage: ao2021_s10_08.py [-h] [-w WIDTH] [-x XINIT] [-y YINIT] file

centroid measurement

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -w WIDTH, --width WIDTH
                        half-width of centroid calculation box (default: 5)
  -x XINIT, --xinit XINIT
                        a rough x coordinate of target
  -y YINIT, --yinit YINIT
                        a rough y coordinate of target

% ./ao2021_s10_08.py -w 10 -x 325 -y 274 synthetic_01.fits
#
# input file name = synthetic_01.fits
# half-width of search box = 10.000000
# x_init = 325.000000
# y_init = 274.000000
#
# result
#  x_centre  = 322.928795
#  y_centre  = 273.562277
#  x_fwhm    = 4.959850
#  y_fwhm    = 5.026942
#  amplitude = 1673.624413
#  theta     = 10.751191
#
# x_fwhm, y_fwhm, fwhm
4.959850 5.026942 4.993396
```

Measured FWHM is ~ 5 pix and it is consistent with the parameter used for generating the synthetic image.

4.2 PSF fitting using 2D Moffat function

Make a Python script to carry out PSF fitting using 2D Moffat function.

Python Code 9: ao2021_s10_09.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy

# importing astropy module
import astropy.io.fits
import astropy.modeling
```

```
# importing photutils module
import photutils.centroids

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# constructing parser object
desc = 'PSF fitting'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
list_psf = ['2dg', '2dm']
parser.add_argument ('-p', '--psf', choices=list_psf, default='2dg', \
                     help='PSF model [2dg=Gaussian, 2dm=Moffat] (default: 2dg)')
parser.add_argument ('-w', '--width', type=int, default=5, \
                     help='half-width of centroid calculation box (default: 5)')
parser.add_argument ('-x', '--xinit', type=int, default=-1, \
                     help='a rough x coordinate of target')
parser.add_argument ('-y', '--yinit', type=int, default=-1, \
                     help='a rough y coordinate of target')
parser.add_argument ('-o', '--output', default='psffitting.png', \
                     help='output file name')
parser.add_argument ('file', nargs=1, default='', help='input file name')

# command-line argument analysis
args = parser.parse_args ()

# input parameters
psf_model   = args.psf
half_width  = args.width
x_init      = args.xinit
y_init      = args.yinit
file_fits   = args.file[0]
file_output = args.output

# checking input file name
if (file_fits == ''):
    print ("You need to specify input file name.")
    sys.exit ()
if not (file_fits[-5:] == '.fits'):
    print ("Input file must be a FITS file.")
    sys.exit ()

# checking output file name
if not ( (file_output[-4:] == '.eps') or (file_output[-4:] == '.pdf') \
         or (file_output[-4:] == '.png') or (file_output[-3:] == '.ps') ):
    print ("Output file must be EPS or PDF or PNG or PS.")
    sys.exit ()

# opening FITS file
hdu_list = astropy.io.fits.open (file_fits)

# reading FITS header
header = hdu_list[0].header

# image size
image_size_x = header['NAXIS1']
```

```
image_size_y = header['NAXIS2']

# checking x_init and y_init
if not ( (x_init >=0) and (x_init < image_size_x) ):
    print ("Input x_init value exceed image size.")
    sys.exit ()
if not ( (y_init >=0) and (y_init < image_size_y) ):
    print ("Input y_init value exceed image size.")
    sys.exit ()

# reading FITS image data
data = hdu_list[0].data

# closing FITS file
hdu_list.close ()

# region of calculation
x_min = x_init - half_width
x_max = x_init + half_width + 1
y_min = y_init - half_width
y_max = y_init + half_width + 1

# extraction of subframe for calculation
subframe = data[y_min:y_max, x_min:x_max]

# rough background subtraction
subframe -= numpy.median (subframe)

# centroid measurement
(xc_com, yc_com) = photutils.centroids.centroid_com (subframe)

# PSF fitting
subframe_y, subframe_x = numpy.indices (subframe.shape)
if (psf_model == '2dg'):
    psf_init = astropy.modeling.models.Gaussian2D (x_mean=xc_com, y_mean=yc_com)
elif (psf_model == '2dm'):
    psf_init = astropy.modeling.models.Moffat2D (x_0=xc_com, y_0=yc_com)
fit = astropy.modeling.fitting.LevMarLSQFitter ()
psf_fitted = fit (psf_init, subframe_x, subframe_y, subframe, maxiter=1000)

# result of fitting
if (psf_model == '2dg'):
    x_centre     = psf_fitted.x_mean.value
    y_centre     = psf_fitted.y_mean.value
    theta        = psf_fitted.theta.value
    x_fwhm       = psf_fitted.x_fwhm
    y_fwhm       = psf_fitted.y_fwhm
    fwhm         = (x_fwhm + y_fwhm) / 2.0
elif (psf_model == '2dm'):
    x_centre     = psf_fitted.x_0.value
    y_centre     = psf_fitted.y_0.value
    alpha        = psf_fitted.alpha.value
    gamma        = psf_fitted.gamma.value
    fwhm         = psf_fitted.fwhm
amplitude    = psf_fitted.amplitude.value
x_centre_sub = x_centre
y_centre_sub = y_centre
x_centre    += x_min
y_centre    += y_min
```

```
# printing information
print ("#")
print ("# input file name = %s" % file_fits)
print ("# half-width of search box = %f" % half_width)
print ("# x_init = %f" % x_init)
print ("# y_init = %f" % y_init)
print ("#")
print ("# result")
print ("#  x_centre  = %f" % x_centre)
print ("#  y_centre  = %f" % y_centre)
print ("#  amplitude = %f" % amplitude)
if (psf_model == '2dg'):
    print ("#  theta     = %f" % theta)
    print ("#  x_fwhm    = %f" % x_fwhm)
    print ("#  y_fwhm    = %f" % y_fwhm)
elif (psf_model == '2dm'):
    print ("#  alpha     = %f" % alpha)
    print ("#  gamma     = %f" % gamma)
    print ("#  fwhm      = %f" % fwhm)
print ("#")

# printing result
print ("# X_CENTRE, Y_CENTRE, FWHM")
print ("%f %f %f" % (x_centre, y_centre, fwhm) )

# making objects "fig" and "ax"
fig = matplotlib.figure.Figure ()
matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax = fig.add_subplot (111)

# axes
ax.set_xlabel ('X [pixel]')
ax.set_ylabel ('Y [pixel]')

# plotting image
im = ax.imshow (subframe, origin='lower')
fig.colorbar (im)
ax.plot (x_centre_sub, y_centre_sub, marker='+', color='red', markersize=10)

# saving file
fig.savefig (file_output, dpi=225)
```

Run the script.

```
% chmod a+x ao2021_s10_09.py
% ./ao2021_s10_09.py -h
usage: ao2021_s10_09.py [-h] [-p {2dg,2dm}] [-w WIDTH] [-x XINIT] [-y YINIT]
                        [-o OUTPUT]
                        file

PSF fitting

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
```

```
  -p {2dg,2dm}, --psf {2dg,2dm}
                        PSF model [2dg=Gaussian, 2dm=Moffat] (default: 2dg)
  -w WIDTH, --width WIDTH
                        half-width of centroid calculation box (default: 5)
  -x XINIT, --xinit XINIT
                        a rough x coordinate of target
  -y YINIT, --yinit YINIT
                        a rough y coordinate of target
  -o OUTPUT, --output OUTPUT
                        output file name

% ./ao2021_s10_09.py -p 2dm -w 10 -x 325 -y 274 -o synthetic_01_moffat.png \
? synthetic_01.fits
#
# input file name = synthetic_01.fits
# half-width of search box = 10.000000
# x_init = 325.000000
# y_init = 274.000000
#
# result
#  x_centre  = 322.928852
#  y_centre  = 273.562137
#  amplitude = 1673.763791
#  alpha     = 309895.412876
#  gamma     = 1669.188207
#  fwhm      = 4.992760
#
# X_CENTRE, Y_CENTRE, FWHM
322.928852 273.562137 4.992760
% ls -l synthetic_01_moffat.png
-rw-r--r--  1 daisuke  taiwan  55301 Apr 16 00:21 synthetic_01_moffat.png
```

Again, we have obtained FWHM of ~ 5 pix as expected.
Show the PNG image. (Fig. 7)

```
% feh -dF synthetic_01_moffat.png
```

5 For your training

1. Read chapter 5 of "Handbook of CCD Astronomy" and learn about image centring and two-dimensional profile fitting.
   - Handbook of CCD Astronomy (2nd Edition)
     - Steve B. Howell
     - Cambridge University Press
     - `https://doi.org/10.1017/CBO9780511807909`

6 Assignment

1. Describe a method to measure the centre of the image of a star. Show mathematical formulae.
2. What is PSF (Point-Spread Function)? Which functions are often used to describe stellar PSF? Show mathematical formulae of those functions.
3. What is FWHM (Full-Width at Half-Maximum)?

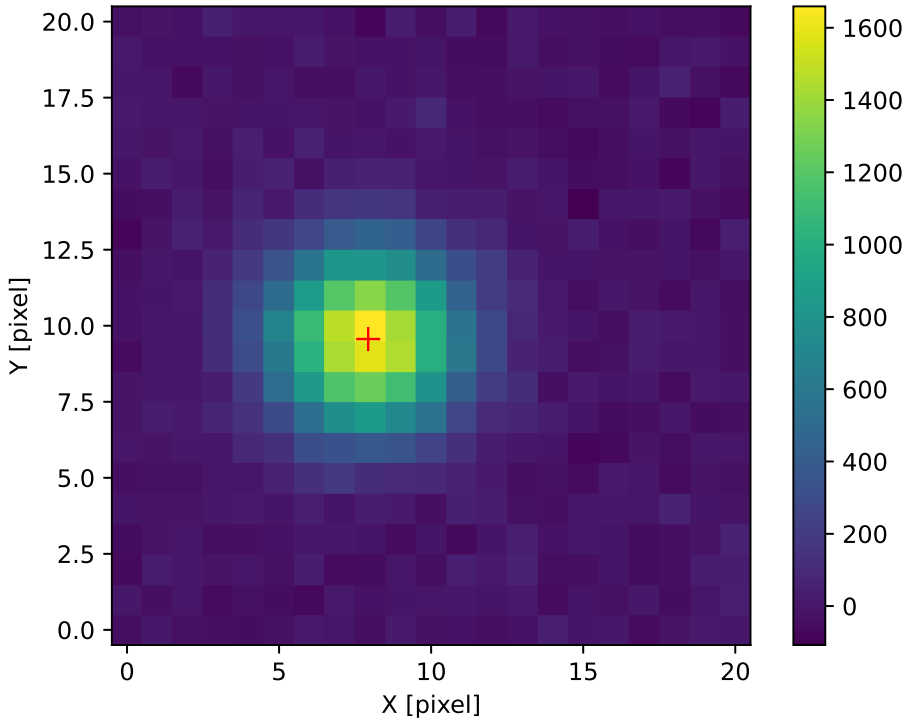

Figure 7: Result of PSF fitting using 2D Moffat function.

4. Consider a Gaussian distribution of the form

$$f_G(x) = \frac{a}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] + b.$$

   Show that FWHM can be written as FWHM $= 2\sqrt{2\log 2}\sigma$.

5. What is Moffat function? Show mathematical formula of 2-dimensional Moffat function.

6. Plot and compare 1-dimensional Gaussian function and 1-dimensional Moffat function.

7. Make your own Python script to produce a synthetic image of a galaxy cluster. Describe the design of your Python script. Show the source code of your Python script. Run the script, and generate a FITS file. Use Ginga to visualise the FITS file and show the image.

8. Select a FITS file of reduced (dark-subtracted and flatfielded) object frame from the session 08 "Basic CCD Data Reduction".

   (a) Which file have you selected?

   (b) Use Ginga to show the image. Choose a star for your measurement. Mark a star with a red circle using Ginga. Export an image. Show the image.

   (c) Make your own Python script to measure centroid of the star. Describe the design of your program. Show the source code of your Python script. Execute the script, and show the result.

   (d) Make your own Python script to carry out PSF fitting of the star. Describe the design of your program. Show the source code of your Python script. Execute the script, and show the result.

9. Make your own Python script to construct a radial profile of a star on the image and carry out fitting using Gaussian function or Moffat function. Describe the design of your program. Show the source code of your Python script. Execute the script, and show the result.